In the Claims:

1. A method for testing a processor using random code generation, the method, comprising:

defining an initial state of the processor, the initial state comprising data locations in the processor, wherein defining the initial state comprises, for each data location:

creating an uncommitted value,

setting an initial state pointer to the uncommitted value, and

setting a current state pointer to the uncommitted value;

generating a random instruction for each data location;

executing the random instructions, the executed random instructions comprising a random program, wherein executing the random instructions produces current values for the data locations, and wherein the current values can be one of committed, uncommitted, and deferred;

determining a length of the random program;

if the random program length equals a desired length, executing a commit V (value) routine; and

if the random program length is less than the desired length, repeating the generating, executing, and determining steps.

2. The method of claim 1, wherein executing the commit V (value) routine, comprises:

determining a state of a value V for each data location;

if the state of the value V is committed, ending the routine as to that value;

if the state of the value V is deferred:

determining an instruction that produced the value V, and

calling an execute instruction routine; and

if the state of the value V is uncommitted:

generating a random number X, and

calling a commit V, X routine.

3. The method of claim 2, wherein calling an execute instruction routine, comprises:

for each input value to an instruction, executing the commit V (value) routine;

performing an instruction set architecture (ISA) operation for the instruction, wherein an ISA result value is generated;

setting an output value of the instruction equal to the ISA result; and

calling a demote instruction routine.

4. The method of claim 3, wherein the instruction is of a type not capable of propagating uncommitted values, and wherein the demote instruction routine comprises marking the output value as deferred.

5. The method of claim 2, wherein calling the commit V, X routine, comprises:

determining if an instruction I produced the value V;

if the instruction I produced the value V:

calling routine reverse propagate I, X, and

executing the instruction I; and

if the instruction I did not produce the value V:

setting the value V equal to X, and

calling a demote V committed routine.

6. The method of claim 5, wherein the instruction I is of an instruction type capable of propagating uncommitted values, wherein the instruction I comprises a single input value, and wherein the reverse propagate I, X routine, comprises:

assigning the value V as an input of the instruction I; and

calling the commit V, X routine.

7. The method of claim 5, wherein the instruction I is of an instruction type capable of propagating uncommitted values, wherein the instruction I comprises an ADD instruction including a plurality of input values, and wherein the reverse propagate I, X routine, comprises:

assigning a value V1 and a value V2 as input of the instruction I;

determining a state of the value V1;

if the state of the value V1 is uncommitted:

calling a commit value V2 routine, and

calling a commit V1, X-V2 routine; and

if the state of the value V1 is not uncommitted:

calling a commit value V1 routine, and

calling a commit V2, X-V1 routine.

8. The method of claim 1, wherein one of the random instructions is a conditional instruction, the method further comprising:

assuming the condition is true, wherein an assumed outcome is determined;

executing the conditional instruction, wherein an actual outcome is produced;

checking the actual outcome against the assumed outcome; and

if the actual and assumed outcomes differ, modifying the instruction to flip a sense of the condition.

9. The method of claim 1, wherein one of the random instructions comprises an immediate operand, the method further comprising, when the immediate operand becomes committed, modifying an opcode of the instruction to include the immediate operand.

10. A method for testing a central processor unit, comprising:

defining data locations in the central processor unit;

assigning random values to each of the data locations, the random values defining an initial state of the central processor unit;

generating random instructions;

executing one or more of the random instructions, wherein the execution provides current values to the data locations; and

determining a current state of the central processor unit, comprising:

marking a value as uncommitted if a content is unknown and any desired content may be created by assigning a content to one or more other uncommitted values,

marking the value as deferred if the content is unknown, and it is not possible to compute any desired content by assigning the content to one or more other uncommitted values, and

marking the value as committed if the content is known.

11. The method of claim 10, further comprising propagating an uncommitted input value to an uncommitted output value, wherein certain instructions cause the propagation.

12. The method of claim 11, further comprising converting an uncommitted value to a committed value with a desirable content, comprising:

locating a predecessor instruction that generated the value to be converted;

using an inverse function of the predecessor instruction to determine a content for an input uncommitted value of the predecessor instruction consistent with the desirable content for the output uncommitted value; and

repeating the locating and determining steps until no predecessor instructions remain.

13. The method of claim 10, wherein one or more instructions include conditions, comprising:

if, when an instruction is generated, one or more data locations that the instruction consumes does not have a known current content, selecting an outcome of the condition; and

if the assignment of initial contents to the plurality of data locations during processing of a first instruction or a later instruction causes an outcome of the condition to differ from an assumed outcome, then modifying the instruction opcode to specify a complementary condition.

14. The method of claim 10, wherein an instruction contains an immediate operand that is consumed by the instruction.

15. The method of claim 14, further comprising:

inhibiting assignment of an initial random number to the immediate operand when the instruction is generated.

16. The method of claim 10, further comprising:

allocating a memory table in a memory, the memory table comprising a plurality of entries; and

generating loads from the memory table to processor registers at random intervals, wherein each entry of the plurality of entries is loaded once.

17. A method for testing a processor using a random code generator, comprising:

defining data locations in the processor;

assigning uncommitted values to each of the defined data locations, wherein an initial state of the processor is defined, and wherein the initial state of the processor is fixed;

generating a random program comprising random instructions, wherein the data locations represent one of input values and output values to the random instructions;

executing the random program;

committing values to desirable values in selected data locations, wherein committed values are produced;

assigning remaining uncommitted values to arbitrary values, wherein a final state of the processor is defined.

18. The method of claim 17, wherein one of the random instructions is a conditional instruction, the method further comprising:

assuming the condition is true, wherein an outcome of the conditional instruction is assumed;

executing the conditional instruction, wherein an actual outcome is produced;

checking the actual outcome against the assumed outcome; and

if the actual and assumed outcomes differ, modifying the instruction to flip a sense of the condition.